Implementing Uniform Reliable Broadcast with Binary Consensus in Systems with Fair-Lossy Links

Jialin Zhang∗
Tsinghua University
zhanggl02@mails.tsinghua.edu.cn

Wei Chen
Microsoft Research Asia
weic@microsoft.com

MSR-TR-2008-162
October 2008

Abstract

When implementing multivalued consensus using binary consensus, previous algorithms assume the availability of uniform reliable broadcast, which is not implementable in systems with fair-lossy links. In this paper, we show that with binary consensus we can implement uniform reliable broadcast directly in systems with fair-lossy links, and thus the separate assumption of the availability of uniform reliable broadcast is not necessary. We further prove that any implementation of uniform reliable broadcast in the fair-lossy link model requires the invocation of an infinite number of binary consensus instances even if no process ever broadcasts any messages, and this is true even when multivalued consensus is used. We also provide a more efficient algorithm implementing uniform reliable broadcast using multivalued consensus. Finally, we prove that both algorithms also satisfy a strong uniform total order property, and thus they actually implement strong uniform atomic broadcast.

Keywords: distributed computing, fault tolerance, binary consensus, uniform reliable broadcast.

1 Introduction

Consensus is a fundamental problem to solve in building fault-tolerant distributed systems. In the consensus problem, each process in the distributed system proposes one value and eventually all processes decide on one of the proposed values, and the decision is irrevocable. The consensus problem characterizes the distributed agreement that is seen in many distributed coordinating tasks such as atomic broadcast, data replication, mutual exclusion, atomic commit, and thus it serves as the basic building block in achieving these tasks.

One basic form of consensus is binary consensus, in which the proposed values are either 0 or 1. Binary consensus is used in studying both impossibility and lower bound results (e.g., [2, 7]) and consensus algorithms (e.g., [1, 4]). However, solving the general multivalued consensus using binary consensus is not trivial. In [10], Turpin and Coan...
provide an algorithm reducing multivalued consensus to binary consensus in synchronous systems with Byzantine failures. In [9], Mostefaoui et al. provide a reduction algorithm in asynchronous systems with crash failures, and in [11] we provide new reduction algorithms that bound the number of invocations to binary consensus instances.

In this paper we resolve one important issue left in [9, 11]. The algorithms in both papers rely on the availability of uniform reliable broadcast (URB) primitives in the system. While URB is implementable in the shared-memory systems and in the message-passing systems with reliable links, it is not implementable in the message-passing systems with fair-lossy links. Informally, a fair-lossy link is one that may drop messages but if a message is sent infinitely often through the link, then the receiver eventually receives it. Fair-lossy link model is more realistic, especially in wide-area networks in which messages do get lost from time to time. In [3], Aguilera et al. show that the weakest failure detector solving URB in systems with fair-lossy links is $\Theta$ (which means that URB is not implementable). Thus, the question is whether URB is implementable in systems with fair-lossy links when binary consensus is also available. In [9, 11], the authors simply assume that in the fair-lossy link model URB is available or equivalently failure detector $\Theta$ is available when solving multivalued consensus using binary consensus.

In this paper, we show that the assumption on the availability of URB or failure detector $\Theta$ is not necessary. Instead, we show that binary consensus can implement URB directly in the fair-lossy link model. People familiar with failure detector researches may notice that solving consensus implies the availability of a quorum failure detector $\Sigma$ ([6]), which is stronger than $\Theta$ ([5]). However, we cannot use the above reasoning directly, because when showing that consensus implies failure detector $\Sigma$, the first step is to show that consensus can implement shared registers, but such implementations require multivalued consensus (to the best of our knowledge). Therefore, in this paper, we provide a direct implementation of URB from binary consensus in the fair-lossy link model. Moreover, we prove a necessity result on any such implementations: any such implementation requires the invocation of an infinite number of binary consensus instances on all correct processes even if no process actually broadcasts any messages, and this result still holds even if we use multivalued consensus instead of binary consensus. Finally, we also provide an algorithm that solves URB using multivalued consensus since it is more efficient than the algorithm using binary consensus. For both algorithms, we show that they actually also satisfy a strong uniform total order property, and thus they implements strong uniform atomic broadcast.

The rest of paper is organized as follows. Section 2 defines the system model and the consensus problem. Section 3 presents the algorithm implementing URB using binary consensus and proves its correctness. Section 4 proves the necessity result, and Section 5 present the algorithm implementing URB using multivalued consensus. We conclude the paper in Section 6.
2 Model and the problem

We consider a message-passing system consisting of \( n \) processes \( \{p_1, p_2, \ldots, p_n\} \). We assume that global time takes non-negative integer values, but it is not accessible to processes. Processes may fail by crashing, i.e., stop taking any actions. A failure pattern \( F \) is a function from a global time \( t \) to a subset of processes that have crashed by time \( t \). A crashed process does not recover, i.e., \( F(t) \subseteq F(t') \) for all \( t \leq t' \). We say that a process \( p \) is faulty (in a failure pattern \( F \)) if it crashes in \( F \) (i.e, there exists a time \( t \) such that \( p \in F(t) \)), and \( p \) is correct if it is not faulty. There is a fair-lossy link between each pair of the processes so that processes can use the links to communicate with each other.

We say a link is fair-lossy if it satisfies the following properties [3]:

- **Fairness**: If a correct process \( p \) sends a message \( m \) to a correct process \( q \) an infinite number of times, then \( q \) eventually receives \( m \) from \( p \).

- **Uniform Integrity**: If \( q \) receives a message \( m \) from \( p \), then \( p \) previously sent \( m \) to \( q \); and if \( q \) receives \( m \) infinitely often from \( p \), then \( p \) sends \( m \) infinitely often to \( q \).

A distributed algorithm \( A \) consists of \( n \) deterministic automata, one for each process. Processes run the algorithm step by step. In one step, one process \( p \) may receive a message \( m \) (or not receiving a message), make a local state transition according to its automaton, its current state, and the message received, and it may send a message to one process. A step is taken at one time point (not accessible to processes), and one process can take at most one step at any time point. A partial run of algorithm \( A \) is a finite sequence of steps that is well-formed, i.e., if \( p \) receives \( m \) from \( q \) in a step, \( q \) must have sent \( m \) to \( p \) in an earlier step. A partial run \( \rho \) is compatible with a failure pattern \( F \) if for each step \( t \), process which do the operation in this step does not contained in \( F(t) \). A run of the algorithm is a sequence of an infinite number of steps together with a failure pattern such that (a) all correct processes take an infinite number of steps; (b) a crashed process does not take any more step after it crashes (according to the failure pattern); and (c) the send and receive primitives on all links satisfy the fair-lossy link properties.

The problem to solve is uniform reliable broadcast (URB) in which a process can broadcasts a value \( v \), which is associated with an attribute \( \text{sender}(v) \) to denote the initiator of the broadcast of \( v \), and eventually correct processes should deliver \( v \). We assume that all values broadcast by all processes are different (e.g., they can be differentiated by process identifiers and local sequence numbers). More precisely, uniform reliable broadcast should satisfy the following properties [8]:

- **Uniform Integrity**: For any value \( v \), any process (correct or faulty) delivers \( v \) at most once, and only if \( v \) was previously broadcast by \( \text{sender}(v) \).
• **Validity**: If a correct process broadcasts $v$, then it eventually delivers $v$.

• **Uniform Agreement**: If a process (correct or faulty) delivers a value $v$, then all correct processes eventually deliver value $v$.

In this paper, we show how to solve uniform reliable broadcast using fair-lossy link and *binary consensus*. The binary consensus is a special form of general consensus in which processes can only propose 0 or 1, and make an irrevocable decision on one value. It needs to satisfy the following three properties:

• **Validity**: If a process decides $v$, then $v$ has been proposed by some process.

• **Uniform Agreement**: No two processes (correct or not) decide differently.

• **Termination**: If all correct processes propose, eventually all correct processes decide.

In our algorithms, we use $FL-Send()$ and $FL-Receive()$ to represent the send and receive primitives on fair-lossy links, $UR-Broadcast()$ and $UR-Deliver()$ to represent uniform reliable broadcast and delivery primitives, and $B-Con()$ to represent a binary consensus instance such that the parameter of $B-Con()$ is the proposal while the return value is the decision value. We use array notation $B-Con[\_]$ to differentiate different binary consensus instances.

### 3 Implementation of URB using binary consensus

We present an algorithm in Figure 1 that solves uniform reliable broadcast using binary consensus instances and fair-lossy links. In the algorithm, the values broadcast by processes are non-negative integers. Any finite-length values can be encoded by a non-negative integer, so using non-negative integers does not lose the generality of the solution.

Each process $p$ maintains two sets $M$ and $D$, where $M$ contains all values submitted for uniform reliable broadcast that $p$ is aware of, and $D$ contains all values that $p$ has $UR-Delivered$. When a process $p$ wants to uniform reliable broadcast a value $v$, it puts $v$ into a set of values $M$ (Task 1). If process $p$ $FL-Receives$ a value $v$ from other processes, $p$ also puts this value $v$ into $M$ (Task 2). Process $p$ periodically $FL-Sends$ all of the values in the set $M \setminus D$ to other processes (Task 3). Process $p$ also periodically runs Task 4, in which it calls binary consensus instances $B-Con[l][0], B-Con[l][1], \ldots, B-Con[l][l]$ one by one, where $l$ is a counter incremented every time the task runs. For each instance $B-Con[l][\_][i], \text{if } i \in M \setminus D$ which means some process has broadcast $i$ but $p$ has not delivered it yet, process $p$ proposes 1 to instance $B-Con[l][\_][i]$ (line 13), otherwise proposes 0 (line 14). If the result of instance $B-Con[l][\_][i]$ is 1, process $p$ $UR-Delivers$ $i$ if $i$ has not been $UR-Delivered$ before and add $i$ into set $D$ (line 17). The following theorem shows that the implementation in Figure 1 satisfies all the properties of uniform reliable broadcast.
Local variables on process $p_i$:
1. $M$, set of known broadcast values, initially empty
2. $D$, set of delivered values, initially empty
3. $l$, non-negative integer, initially 0
4. $r$, 0 or 1, representing the result of current binary consensus instance

Code for process $p_i$:
5. Task 1: To execute $UR$-$Broadcast(v)$:
   6. $M \leftarrow M \cup \{v\}$
7. Task 2: Upon $FL$-$Receive(v)$:
   8. $M \leftarrow M \cup \{v\}$
9. Task 3: Repeat periodically:
   10. for any value $v \in M \setminus D$, $FL$-$Send(v)$ to all processes
11. Task 4: Repeat periodically:
   12. for $i \leftarrow 0$ to $l$ do
   13. if $i \in M \setminus D$ then $r \leftarrow B-Con[l][i](1)$
   14. else $r \leftarrow B-Con[l][i](0)$
   15. if $(r = 1) \text{ and } (i \notin D)$
   16. $D \leftarrow D \cup \{i\}$
   17. $UR$-$Deliver(i)$
   18. endfor
   19. $l \leftarrow l + 1$

Figure 1: Implementation of uniform reliable broadcast using binary consensus.

Theorem 1 The algorithm in Figure 1 implements uniform reliable broadcast using binary consensus instances in a system with fair-lossy links.

Proof. We first prove that no process will be blocked in the algorithm. All processes run the binary consensus instance $B-Con[i][j]$ in the same order: $B-Con[0][0]$, $B-Con[1][0]$, $B-Con[1][1]$, $B-Con[2][0]$, $B-Con[2][1]$, $B-Con[2][2]$, ... So for any binary consensus instance $B-Con[i][j] (i \geq j)$, all correct processes will eventually propose to it. By the Termination property of binary consensus, all correct processes will eventually decide in this instance. Therefore, no correct process will be blocked anywhere in the algorithm. Next we show that the algorithm satisfies the Uniform Agreement, Uniform Integrity, and Validity properties of uniform reliable broadcast.

Uniform Agreement: If a process $p$ $UR$-Delivers value $v$, it must have a binary consensus instance $B-Con[l][v]$ that returns 1 for some $l$. For any correct process $q$, when it runs procedure from line 12–18 with $l$, the return value of $B-Con[l][v]$ on process $q$ must be 1, according to the Uniform Agreement property of binary consensus. Thus, if $q$ has not $UR$-Delivered value $v$ before, it will $UR$-Deliver it in line 17. So Uniform Agreement property holds.

Uniform Integrity: When process $p$ $UR$-Delivers value $v$, we have $v \in D$ by line 16. Then, by the second condition in line 15, process $p$ will never deliver it again. So any process can $UR$-Deliver any value at most once. If process $p$ $UR$-Delivers value $v$ in line 17, then the binary consensus instance $B-Con[l][v]$ returns 1 for some $l$. So some process $q$ proposes 1 to this instance. The only case to propose 1 is in line 13 which means $v \in M \setminus D$ in process $q$ at that time. By the Uniform Integrity property of fair-lossy links, we know that all values in $M$ are previously broadcast by some process. This proves the Uniform Integrity property.
**Validity:** We prove it by contradiction. Suppose value $v$ is UR-Broadcast by some correct process $p$ but never UR-Delivered by $p$. Since we have proven the Uniform Agreement property of the uniform reliable broadcast algorithm, we know that no process ever UR-Deliver $v$ in the run. Since $v$ is never UR-Delivered by process $p$, $v$ is permanently in $M \setminus D$ on process $p$ after $p$ UR-Broadcasts it. Thus, process $p$ will FL-Send($v$) to all other processes infinitely often by line 10. By the Fairness property of fair-lossy links, every correct process $q$ will eventually FL-Receive value $v$, so that $v \in M$ on process $q$ after that time. Since no process will ever UR-Delivered $v$, $v$ will never be in set $D$ on any process. So there exists a time point after which all correct processes have $v \in M \setminus D$ and all faulty processes have crashed. At that time point, suppose $l \geq v$ is the smallest integer that binary consensus $B-Con[l][v]$ has never been called by any process. Then, for any correct process $p$, when it runs binary consensus instances $B-Con[l][v]$, it will propose 1 by line 13. Thus, $B-Con[l][v]$ must return 1 by the Validity property of binary consensus. Therefore, $p$ will UR-Deliver $v$ by line 15 and 17. This contradicts that $v$ is never UR-Delivered by any process. So Validity holds.

Theorem 1 shows that the algorithm in Figure 1 implements uniform reliable broadcast. This result together with the results in [9, 11] is enough to show that using binary consensus instances alone can solve multivalued consensus in systems with fair-lossy links. However, the above algorithm actually implements a stronger specification, namely strong uniform atomic broadcast, because it satisfies the following Strong Uniform Total Order property:

- **Strong Uniform Total Order:** If process $p$ (correct or faulty) delivers two values $v$ and $v'$ in this order, for any process $q$ (correct or faulty) that delivers $v'$, $q$ must have delivered $v$ before $v'$.

We say that a broadcast is a strong uniform atomic broadcast if it is uniform reliable broadcast and it also satisfies the Strong Uniform Total Order property.

**Lemma 1** The algorithm in Figure 1 satisfies the Strong Uniform Total Order property.

**Proof.** Suppose process $p$ (correct or faulty) delivers two values $v$ and $v'$ in this order, consider any process $q$ (correct or faulty) that delivers $v'$. Note that any process can deliver value $v$ only after some binary consensus instance $B-Con[l][v]$ returns 1. Let $l$ (or $l'$) be the minimum integer satisfying (1) $l \geq v$ (or $l' \geq v'$) and (2) $B-Con[l][v]$ (or $B-Con[l'][v']$) returns 1. All processes run the binary consensus instance $B-Con[v][j]$, the same order: $B-Con[0][0], B-Con[1][0], B-Con[1][1], B-Con[2][0], B-Con[2][1], B-Con[2][2], \cdots$, so we have (1) $l' > l$ or (2) $l' = l$ and $v' > v$, because $p$ delivers $v$ before $v'$. Since process $q$ delivers value $v'$, $q$ must have invoked binary consensus instance $B-Con[l'][v']$.

---

1 This property is stronger than the Uniform Total Order property given in [8]. The property in [8] requires ordering only when both $p$ and $q$ delivers $v$ and $v'$, but our property here requires ordering when $p$ delivers $v$ and $v'$ while $q$ delivers $v'$. For example, the uniform atomic broadcast in [8] allows a run in which a correct process $p$ delivers $v$ and $v'$ while a faulty process $q$ only delivers $v'$, but our strong uniform atomic broadcast specification does not allow this run.
Thus process $q$ must have invoked binary consensus instance $B$-$Con[l][v]$ before and get result 1 from this instance. Then process $q$ delivers value $v$ before it delivers value $v'$. $\square$

From Theorem 1 and Lemma 1, we have

**Theorem 2** The algorithm in Figure 1 implements strong uniform atomic broadcast using binary consensus instances in a system with fair-lossy links.

A direct consequence of Theorem 2 is that we can use the algorithm in Figure 1 to implement multivalued consensus directly instead of using it to implement uniform reliable broadcast first and then using the algorithm in [9] or [11] to implement multivalued consensus. With strong uniform atomic broadcast, it is straightforward to implement multivalued (uniform) consensus. The algorithm is exactly the same as the one in Figure 13 of [8]: to propose $v$, $p$ broadcasts $v$, and when $p$ delivers the first value $v'$, $p$ decides $v'$. It is easy to show that this algorithm is correct when there is at least one correct process in any run.\(^2\)

## 4 Necessary condition to implement URB

In the previous section, we give an algorithm which uses the binary consensus instances to implement uniform reliable broadcast. One issue of the algorithm is that each process needs to call an infinite number of binary consensus instances, even if the number of the values broadcast by all processes is finite (or even zero). In this section, we show that this is necessary. Moreover, its reason is not that binary consensus is not as powerful as multivalued consensus, but because the fair-lossy link model is strictly weaker than the reliable link model. In the following theorem, we prove that even if we allow multivalued consensus instances in our algorithm, the requirement of infinite consensus instances is still necessary. Note that we assume that consensus instances as black boxes, which means that uniform reliable broadcast algorithms can only access these instances through their interfaces (proposing a value and receiving a decision), and algorithms cannot access or modify the implementations of consensus (e.g., piggybacking messages onto messages in the implementation of consensus).

**Theorem 3** In the fair-lossy model, for any algorithm $A$ that implements uniform reliable broadcast using multivalued consensus instances as black boxes, for any partial run $\rho$ of $A$ and any failure pattern $F$ compatible with $\rho$, there exists a run $R$ with failure pattern $F$ such that (1) $\rho$ is the initial sequence of $R$; (2) no process broadcasts any value after $\rho$; and (3) all correct processes invoke multivalued consensus instances an infinite number of times.

---

\(^2\)The algorithm would not be correct if we use the uniform atomic broadcast specification in [8]. For example, the sample run given in Footnote 1 is allowed with uniform atomic broadcast, but it causes $p$ to decide $v$ and $q$ to decide $v'$, violating the Uniform Agreement property of consensus.
The theorem means that no matter how the system is currently running, it is always possible that all correct processes will invoke an infinite number of consensus instances even if no process broadcast any more value from now on. In the special case when \( \rho \) is the empty sequence, the theorem shows that for any failure pattern \( F \), there exists a run with \( F \) in which no process ever broadcasts any value but all correct processes invoke infinitely many consensus instances. In order to prove this theorem, we first prove the following lemma. We say that a partial run \( \rho' \) is an extension of a partial run \( \rho \) if \( \rho \) (as a sequence of steps) is a prefix of \( \rho' \), and we write \( \rho' = \rho \cdot \rho'' \) where \( \rho'' \) is the sequence of additional steps after \( \rho \).

**Lemma 2** In the fair-lossy model, for any algorithm \( A \) that implements uniform reliable broadcast using multivalued consensus instances as black boxes, for any partial run \( \rho \) and any process \( p \), there exists a partial run \( \rho' \) such that (1) \( \rho' = \rho \cdot \rho'' \) is an extension of \( \rho \); (2) no process broadcasts any value in \( \rho'' \); and (3) \( p \) invokes at least one multivalued consensus instance in \( \rho'' \).

**Proof.** Suppose, for a contradiction, that there exists a partial run \( \rho \) and a process \( p \), for all the partial extensions \( \rho' = \rho \cdot \rho'' \) of \( \rho \) in which no process broadcasts any value in \( \rho'' \), \( p \) does not invoke any multivalued consensus instance in \( \rho'' \). Suppose that the last step of \( \rho \) is executed at time \( t \).

Consider a full run \( R_1 \) such that the following properties hold: (1) The failure pattern is such that \( p \) is correct and all other processes crash at time \( t + 1 \); (2) the sequence of steps by time \( t \) is exactly \( \rho \); and (3) after time \( t \), \( p \) does not broadcast any values. In \( R_1 \), \( p \) does not invoke any multivalued consensus instance after \( t \), because otherwise, there would be a partial run extension of \( \rho \) in which no process broadcasts any value after \( \rho \) but \( p \) invokes consensus after \( \rho \).

Let \( q \) be a process different from \( p \). We now construct run \( R_2 \) such that the following properties hold: (1) The failure pattern of run \( R_2 \) is such that \( q \) is correct and all other processes crash at time \( t + 1 \); (2) the sequence of steps by time \( t \) is exactly \( \rho \); and (3) at time \( t + 1 \), process \( q \) broadcasts a value \( v \) that is different from any values broadcast in partial run \( \rho \). Since \( q \) is a correct process, by the Validity property of uniform reliable broadcast, \( q \) eventually delivers \( v \) in \( R_2 \). Suppose, process \( q \) delivers value \( v \) at time \( t' > t \).

Now, we construct the run \( R_3 \) as follows. The failure pattern of \( R_3 \) is such that \( p \) is correct, \( q \) crashes at time \( t' + 1 \) and all other processes crash at time \( t + 1 \). The sequence of steps by time \( t \) is exactly \( \rho \). From time \( t + 1 \) to \( t' \), \( p \) does not take any step and \( q \) takes exactly the same steps as in \( R_2 \). Thus, process \( q \) cannot distinguish run \( R_2 \) and \( R_3 \) up to time \( t' \), and \( q \) delivers value \( v \) at time \( t' \). At time \( t' + 1 \), process \( q \) crashes. After time \( t' \), \( p \) takes an infinite number of steps, but it does not broadcasts any value, and it no longer receives any messages from other processes, which means that all messages that are sent to process \( p \) but is not received by \( p \) by time \( t \) are lost. This does not violate the fair-lossy link specification because all processes other than \( p \) are faulty, and there are only a finite number of messages sent.
by these processes to $p$ before they crash. Since $p$ does not receive any messages and does not broadcast any value after time $t$, in $R_3$ $p$ must follow the same sequence of steps as in $R_1$. Since in $R_1$ $p$ does not invoke any consensus instance after time $t$, $p$ cannot distinguish run $R_1$ and $R_3$ and thus $p$ would never deliver value $v$ (otherwise, $p$ could invoke a consensus instance that is also invoked by $q$ before $q$ crashes such that the information about $q$ delivering $v$ could be conveyed through the consensus instance). This violates the Uniform Agreement property of uniform reliable broadcast. Therefore, the lemma holds.

The lemma illustrates a fact that even if there is no events triggering a process (i.e. receiving some message or receiving a broadcast invocation), the process should still call consensus instances. The intuition is that in the fair-lossy model the system may drop an arbitrary number of messages from a faulty process but uniform reliable broadcast still requires a correct process to deliver all values delivered by faulty processes, so the only “reliable way” for the algorithm to transfer information from faulty processes to correct processes is through (uniform) consensus instances.

We now prove Theorem 3.

**Proof of Theorem 3.** Consider an arbitrary partial run $\rho$ and a failure pattern $F$ compatible with $\rho$. We first extend $\rho$ such that (1) no process broadcasts any more values; (2) all correct processes take steps in the round-robin fashion, and they receive messages sent to them in the FIFO order as long as there is some message not received yet; and (3) all faulty processes do not take step until they crash at the time points indicated by $F$. We continue extending $\rho$ in this way until all correct processes deliver all values they are supposed to deliver according to the uniform reliable broadcast specification, i.e., they deliver all values that are either broadcast by correct processes or have delivered by faulty processes in $\rho$. Since algorithm $A$ implements uniform reliable broadcast, we can find a partial run $\rho_0$ such that $\rho_0$ is an extension of $\rho$ and all values that should be delivered have been delivered in $\rho_0$.

Let $p_1, p_2, \ldots, p_k$ be the correct processes. We construct the run $R$ as follows starting from $\rho_0$. By Lemma 2, there exists a partial run extension $\rho'_0 = \rho_0 \cdot \rho''_0$ such that no process broadcasts in $\rho''_0$ and $p$ invokes at least one multivalued consensus instance in $\rho''_0$. After $\rho'_0$, we further extend it by letting every correct process take at least one step and all correct processes receive all messages sent to them during $\rho'_0$. This can be done with a finite number of steps and we obtain a further partial run extension $\rho_1$. We then apply Lemma 2 on $\rho_1$ and $p_2$ to obtain an extension $\rho'_1$ in which $p_2$ invokes at least one consensus instance after $\rho_1$, and then let all correct processes take steps and receive all messages sent in $\rho'_1$. In general, we can repeat this procedure infinitely many times, applying Lemma 2 on the new partial run and every correct process in the round-robin fashion. We thus obtain a sequence with an infinite number of steps. In this sequence, all correct processes take an infinite number of steps, and they receive all messages sent to them. Therefore, together with the failure pattern $F$, it forms a run $R$ of the algorithm. In $R$, $\rho$ is the initial sequence.
of \( R \), no process ever broadcasts any value after \( \rho \), and all correct processes invoke an infinite number of multivalued consensus instances by our construction. Therefore, the theorem holds.

Several remarks are now in order on the subtlety of the theorem. First, we treat the consensus instances as black boxes, so that we cannot use the messages sent in the underlying implementation of consensus to transfer any information for the implementation of uniform reliable broadcast. Second, from Lemma 2 we see that the key reason establishing the proof is that a correct process \( p \) cannot know if a faulty process \( q \) delivers a value \( v \) broadcast by some faulty process unless it calls consensus instances. One implication is that even if all correct processes know exactly that there will be no broadcast requests in the future, they still need to keep calling consensus instances to see if any faulty process delivers any value. Thus the theorem is strong in this sense. Finally, we consider a system model in which an arbitrary number of processes may fail (i.e., the wait-free model). If we consider a system in which a majority of processes are correct in all runs, we can implement uniform reliable broadcast without using consensus instances.

5 Implementation of URB using multivalued consensus

Theorem 3 shows that even with multivalued consensus, processes still need to call consensus instances infinitely often. However, using multivalued consensus does make the implementation of uniform reliable broadcast simpler and more efficient than using binary consensus. In Figure 2 we present an algorithm that implements uniform reliable broadcast in the fair-lossy model using multivalued consensus instances, which is represented as \( M-\text{Con}() \) in the algorithm. As the case of binary consensus, the algorithm in Figure 2 actually implements strong uniform atomic broadcast. The structure of the algorithm is very close to the algorithm of Figure 14 in [8], which implements atomic broadcast using consensus and reliable broadcast. The main differences are: (1) processes directly use send and receive primitives instead of (nonuniform) reliable broadcast; and (2) processes need to periodically invoke consensus instances, while in [8] processes invoke consensus only when there are new messages received. As the result, in [8] processes invoke at most \( k \) consensus instances if there are \( k \) broadcast requests, while in our algorithm processes need to invoke an infinite number of consensus instances even if there is no broadcast requests, as shown by Theorem 3.

In the algorithm, each process \( p \) periodically invokes consensus instances with the set of all values that it has received but not yet delivered \( (M \setminus D) \) as the proposals to the consensus instances (line 12). When it receives the return value (a set), it \( \text{UR-Delivers} \) all values in the set that has not been \( \text{UR-Delivered} \) before in a predefined deterministic order (line 14).
Local variables on process $p_i$:
1. $M$, set of known broadcast values, initially empty
2. $D$, set of delivered values, initially empty
3. $R$, set of values representing the result of current multivalued consensus instance
4. $l$, non-negative integer, initially 0

Code for process $p_i$:

5. Task 1: To execute $UR$-Broadcast($v$):
6. $M \leftarrow M \cup v$

7. Task 2: Upon $FL$-Receive($v$):
8. $M \leftarrow M \cup v$

9. Task 3: Repeat periodically:
10. for any value $v \in M \setminus D$, $FL$-Send($v$) to all processes

11. Task 4: Repeat periodically:
12. $R \leftarrow M$-Con$l([M \setminus D])$
13. $R \leftarrow R \setminus D$
14. $UR$-Deliver all values in $R$ in some deterministic order
15. $l \leftarrow l + 1$

Figure 2: Implementation of uniform reliable broadcast using multivalued consensus.

Theorem 4 The algorithm in Figure 2 implements strong uniform atomic broadcast using multivalued consensus instances in a system with fair-lossy links.

Proof (Sketch). The proof is almost the same as the original proof in [8]. We only sketch the proof on Validity and Strong Uniform Total Order properties here.

For Validity, suppose for a contradiction that correct process $p$ broadcasts a value $v$ but never delivers it. Then value $v$ is in $M$ and never in $D$ in process $p$ after $p$ broadcasts it. Moreover, by the Uniform Agreement property of uniform reliable broadcast (can be proven independently as in [8]), no process ever delivers $v$, and thus $v$ is never in $D$ on any process. Thus, process $p$ will $FL$-Send $v$ to all other processes infinitely often by line 10. By the Fairness property of fair-lossy link, any correct processes $q$ will eventually $FL$-Receive $v$, so that $v \in M$ on process $q$ after that time. So there exists a time point after which all faulty processes have crashed and all correct processes permanently have $v \in M \setminus D$. After this time point, the proposals of all processes to any consensus instances contain value $v$. Then, by the Validity property of consensus, any decision of any such consensus instances contains $v$, and thus all correct processes will deliver $v$ by line 13 and 14. This contradiction concludes the proof of Validity property.

For Strong Uniform Total Order, suppose that $p$ delivers $v$ and $v'$ in this order and $q$ delivers $v'$. We first claim that for every value $v$ that a process $p$ delivers, $v$ is in the decision set of exactly one consensus instance. If not, suppose $v$ appears in the decision set of two consensus instances $l$ and $l'$ with $l < l'$. By the Validity of consensus, some process $p'$ proposes a set containing $v$ to instance $l'$. Then $p'$ must have completed instance $l$, and thus $p'$ must have delivered $v$ before it proposes to instance $l'$. But if so, $v$ is in $D$ on $p'$ and thus $p'$ will not propose a set containing $v$ to instance $l'$, a contradiction. With the claim, let $l$ and $l'$ be consensus instance numbers when $p$ delivers $v$ and $v'$. So we have
\( l \leq l' \) since \( v \) is delivered before \( v' \). Since \( q \) also delivers \( v' \), by the Uniform Agreement property of consensus, \( q \) also delivers \( v' \) after completing consensus instance \( l' \). If \( l < l' \), then \( q \) must have delivered \( v \) after completing instance \( l \). If \( l = l' \), then \( v \) must be in the decision set of consensus instance \( l' \), and since \( p \) and \( q \) follow the same deterministic order in delivering values, \( q \) must have delivered \( v \) before delivering \( v' \). Therefore, Strong Uniform Total Order holds.

\[ \square \]

6 Conclusion

In this paper, we show that uniform reliable broadcast can be implemented in systems with fair-lossy links when binary consensus is available, and thus the separate assumption on the availability of uniform reliable broadcast or an equivalent failure detector \( \Theta \) is unnecessary when implementing multivalued consensus from binary consensus. In our algorithm, every process needs to invoke binary consensus periodically even if there is no message being broadcast, and we prove that this behavior is inevitable.

With this work, we can finally claim that binary consensus indeed has the same power in terms of solvability as multivalued consensus in systems with fair-lossy links. We can then apply results obtained with multivalued consensus case to binary consensus. For example, the weakest failure detector for binary consensus is the same as the weakest failure detector for multivalued consensus in this model.

References


